In the Office Action, the Examiner rejected claims 1-4, 6-9, 12-16, 18-21, and 24 under 35 U.S.C. § 102(e) as being anticipated by McAllister et al. (United States Patent Number 6,215,765) and rejected claims 5, 10, 11, 17, 22 and 23 under 35 U.S.C. §103(a) as being unpatentable over McAllister et al., and further in view of Ash et al. (United States Patent Number 4,345,116).

The present invention is directed to a method and apparatus for alleviating congestion and overload in a distributed call-processing system interconnected through a packet based network, such as an IP or an ATM network. The illustrative IP network includes a plurality of end terminals (ETs) and distributed call processors (CPs). When an end terminal (ET) wants to place a call, the end terminal (ET) send a call set up message to a call processor (CP). According to an aspect of the invention, the call processor will determine whether to process the request or to forward the request to another call processor. Generally, the call processor will declare an overload condition if sufficient resources (such as processing or memory resources) are not available to process a given call. If a call processor determines that it is too congested to process a call, the call processor enters an overload condition, selects an alternate call processor and forwards the request to the alternate call processor.

A substitute specification is included herewith to correct typographical errors.

Independent Claims 1, 8, 13 and 20

Independent claims 1, 8, 13, and 20 were rejected under 35 U.S.C. § 102(e) as being anticipated by McAllister et al.

Regarding claims 1 and 8, the Examiner asserts that McAllister teaches that a "call processor is congested."

Applicants note that McAllister is directed to rerouting a call due to congestion or physical failure. See, Abstract. McAllister defines congestion in regard to network links, not call processors. McAllister teaches that "congestion may occur on a network link if many incoming streams of traffic all terminate on the same outbound link, or the outbound link may (be) busy or down due to failure." Col. 1, lines 10-12. Independent claims 1 and 13 require "whereby said forwarded call set up request indicates to said alternate call processor that said congested call

processor is congested" and independent claims 8 and 20 require "setting a flag associated with said congested call processor."

Thus, McAllister does not disclose or suggest "whereby said forwarded call set up request indicates to said alternate call processor that said congested call processor is congested," as required by independent claims 1 and 13, and does not disclose or suggest "setting a flag associated with said congested call processor," as required by independent claims 8 and 20.

Additional Cited References

Ash et al. was also cited by the Examiner in rejecting claims 5, 10, 11, 17, 22, and 23 for its disclosure that Ash teaches that "crankback is used in a time sensitive environment where alternate routing is responsive to variations in traffic demand."

Applicant notes that Ash is directed to an "alternate routing method which allows route choices without regard to network hierarchy. A plurality of routing sequences is generated, each route sequence including a plurality of route choices and being time sensitive to traffic demands, subject to a grade of service constraint and used for some predetermined time interval during which the sequence tends to mitigate network cost." See, Abstract. Ash does not address the issue of handling congested call processors.

Thus, Ash does not disclose or suggest "whereby said forwarded call set up request indicates to said alternate call processor that said congested call processor is congested," as required by independent claims 1 and 13, and does not disclose or suggest "setting a flag associated with said congested call processor," as required by independent claims 8 and 20.

Dependent Claims 2-7, 9-12, 14-19 and 21-24

Dependent claims 2-4, 6-7, 9, 12, 14-16, 18-19, 21, and 24 were rejected under 35 U.S.C. § 102(e) as being anticipated by McAllister et al. and dependent claims 5, 10, 11, 17, 22 and 23 were rejected under 35 U.S.C. §103(a) as being unpatentable over McAllister et al., and further in view of Ash et al.

Claims 2-7, 9-12, 14-19, and 21-24 are dependent on claims 1, 8, 13, and 20, respectively, and are therefore patentably distinguished over McAllister et al. and Ash et al. (alone or in any combination) because of their dependency from amended independent claims 1, 8, 13, and 20

for the reasons set forth above, as well as other elements these claims add in combination to their base claim.

If any outstanding issues remain, or if the Examiner has any further suggestions for expediting allowance of this application, the Examiner is invited to contact the undersigned at the telephone number indicated below.

The Examiner's attention to this matter is appreciated.

Respectfully submitted,

Date: July 3, 2003

Kevin M. Mason
Attorney for Applicant(s)
Reg. No. 36,597
Ryan, Mason & Lewis, LLP
1300 Post Road, Suite 205
Fairfield, CT 06824
(203) 255-6560

VERSION MARKED TO SHOW ALL CHANGES

IN THE ABSTRACT:

A method and apparatus are disclosed for alleviating congestion and overload in a distributed call-processing system interconnected through a packet based network, such as an Internet Protocol [IP] or an Asynchronous Transfer Mode [ATM] network. The illustrative Internet Protocol [IP] network includes a plurality of end terminals [(ETs)] and distributed call processors [(CPs)]. When an end terminal [(ET)] wants to place a call, the end terminal [(ET)] send a call set up message to a call processor [(CP)]. According to an aspect of the invention, the call processor will determine whether to process the request or to forward the request to another call processor. Generally, the call processor will declare an overload condition if sufficient resources (such as processing or memory resources) are not available to process a given call. If a call processor determines that it is too congested to process a call, the call processor enters an overload condition, selects an alternate call processor and forwards the request to the alternate call processor. A given call processor [(CP)] implicitly announces its overload condition to another call processor by virtue of the forwarded congestion message. Each call processor [(CP)] maintains an ordered list of call processors that indicates whether or not each call processor [(CP)] is overloaded. The present invention attempts to distribute forwarded congestion messages among all of the available alternate call processors [(CPs)], for example, using a last message sent [(LMS)] flag. Generally, a call processor [(CP)] in an overload condition will not forward another congestion message to a call processor [(CP)] having its last message sent [(LMS)] flag set unless there are no other call processors [(CPs)] available. The congested call processor attaches a call processor identifier to the forwarded congestion message, indicating to the recipient call processor that the congested call processor is in an overload condition. Thus, a forwarded congestion message will cause the recipient call processor to set a flag, for example, in the ordered list of call processors [(CPs)], indicating that the congested call processor is congested. In one embodiment, each congestion flag has an associated timer that causes the flag to expire (or reset) after a predefined time interval that permits the congested call processor to recover from the overload condition.

Marked Up Specification

METHOD AND APPARATUS FOR MESSAGE-BASED OVERLOAD CONTROL IN A DISTRIBUTED CALL-PROCESSOR COMMUNICATION SYSTEM

Field of the Invention

The present invention relates to packet communication systems, and more particularly, to method and apparatus for congestion management in a distributed call-processor communication system.

Background of the Invention

Communication networks are used to transfer information, such as data, voice, text or video information, among communication devices, such as packet telephones, computer terminals, multimedia workstations, and videophones, connected to the networks. A network typically comprises nodes connected to each other, and to communication devices, by various links. Transmitted information may be of any form, but is often formatted into packets or cells.

Packet-switching network architectures, such as networks using Internet Protocol (IP) or asynchronous transfer mode (ATM) protocols, are widely used. In a packet-switched network, data transmissions are typically divided into blocks of data, called packets, for transmission through the network. For a packet to get to its proper destination, the packet must traverse through one or more network switches, routers or intermediate systems. Typically, a packet includes a header, containing source and destination address information, as well as a payload (the actual application data).

When a call is initiated in an Internet Protocol [IP] network environment, a call processor performs the required tasks to setup the call and allocate the necessary resources. In such an environment, a congestion management policy is required to ensure that sufficient network resources are available in the network to handle the signaling and control of the call. If the call processor is in an "overload" condition, where the volume of signaling traffic exceeds the capacity of the call processor, the call processor should exercise overload control. If overload is not properly controlled, system throughput can be reduced, and even cause the network to cease operation. In order to effectively control the load, many systems drop the incoming call requests in order to preserve the quality of service for the ongoing calls. However, in a distributed

environment, a better policy is to identify an alternate processor that can handle the new call. If such an alternate processor cannot be found, then the new call is dropped.

Currently, many communication systems rely on a distributed call-processing architecture for reliability and scalability reasons. Internet Protocol[IP]-based private branch exchange (IP-PBX) switches, for example, distribute the call processing functionality among many servers. Thus, while the initial call processor that receives the call admission request may be in an overload condition, another call processor in the distributed network environment may be available to process the call.

A number of congestion management techniques have been proposed or suggested that determine the availability of an alternate call processor. These congestion management techniques generally rely on periodic polling of the other call processors in the distributed network Typically, each call processor communicates with every other call processor in the distributed network environment to collect statistics for each call processor. The collected statistics help determine the availability of each call processor to perform a specific task in the event of an overload condition. Thus, such polling-based congestion management techniques increase network overhead and potentially contribute to the overload conditions they are attempting to mitigate.

As apparent from the above-described deficiencies with conventional systems for overload control, a need exists for an improved method and apparatus for overload control in a distributed network environment that admits as many calls as possible. A further need exists for an overload control method and apparatus that alleviate congestion and control overload in a distributed call-processing system with minimal overhead and a low processing requirement load by the call processors.

Summary of the Invention

Generally, a method and apparatus are disclosed for alleviating congestion and overload in an Internet Protocol [IP-]network having a distributed call-processing system. The illustrative Internet Protocol [IP] network includes a plurality of end terminals (ETs) and distributed call processors (CPs). When an end terminal [(ET)] wants to place a call, the end terminal [(ET)] sends a call set up message to a call processor [(CP)]. According to an aspect of

the invention, the call processor will determine whether to process the request or to forward the request to another call processor. Generally, the call processor will declare an overload condition if sufficient resources are not available to process a given call.

According to an aspect of the invention, if a call processor determines that it is too congested to process a call, the call processor enters an overload condition, selects an alternate call processor and forwards the request to the alternate call processor. A given call processor [(CP)] implicitly announces its overload condition to another call processor by virtue of the forwarded call setup request message. According to another feature of the invention, each call processor [(CP)] maintains an ordered list of call processors that indicates whether or not each call processor [(CP)] is overloaded in addition to providing a preferred list of call processors to handle the overflow traffic. In this manner, an alternate call processor [(CP)] can be selected using the ordered list of call processors. The present invention will result in distributing the forwarded call setup request messages, carrying the congestion indication among all of the available alternate call processors [(CPs)]. In one implementation, a last message sent (LMS) flag is utilized that indicates the last call processor to receive a forwarded congestion message. Generally, a call processor [(CP)] in an overload condition will not forward another congestion message to a call processor [(CP)] having its last message sent [(LMS)] flag set unless there are no other call processors [(CPs)] available.

According to another aspect of the invention, the congested call processor attaches a call processor identifier to the forwarded congestion message, indicating to the recipient call processor that the congested call processor is in an overload condition. Thus, a forwarded congestion message will cause the recipient call processor to set a flag, for example, in the ordered list of call processors [(CPs)], indicating that the congested call processor is congested. In one embodiment, each congestion flag has an associated timer that causes the flag to expire (or reset) after a predefined time interval that permits the congested call processor to recover from the overload condition.

A more complete understanding of the present invention, as well as further features and advantages of the present invention, will be obtained by reference to the following detailed description and drawings.

Brief Description of the Drawings

FIG. 1 illustrates a network environment in which the present invention can operate;

FIG. 2 is a schematic block diagram of a call processor of FIG. 1 in accordance with the present invention;

FIG. 3 is a sample table from the overload control analysis table of FIG. 2;

FIG. 4 is a flow chart describing an exemplary outgoing congestion evaluation process incorporating features of the present invention and employed by the call processor of FIG. 2;

FIG. 5 is a flow chart describing an exemplary incoming congestion evaluation process incorporating features of the present invention and employed by the call processor of FIG. 2;

FIGS. 6A and 6B illustrate the overload control analysis table of FIG. 3 populated with data for call processor (CP1) of FIG. 1 before and after, respectively, detecting an overload condition and forwarding a call set up message to an alternate call processor;

FIGS. 7A and 7B illustrate the overload control analysis table of FIG. 3 populated with data for call processor (CP4) before and after, respectively, receiving a forwarded call set up message from call processor (CP1); and

FIGS. 8A and 8B illustrate the overload control analysis table of FIG. 3 populated with data for call processor (CP1) of FIG. 1 to illustrate the use of the last message sent [(LMS)] flag bit in accordance with one feature of the present invention.

Detailed Description

FIG. 1 illustrates a network environment in which the present invention can operate. As shown in FIG. 1, the network 100 interconnects a plurality of end terminals [(ETs)] 110-1 through 110-K (hereinafter, collectively referred to as end terminals [(ETs)] 110) and call processors [(CPs)] 120-1 through 120-N (hereinafter, collectively referred to as call processors (CPa) 120). The network 100 may be embodied as any IP or data network infrastructure and generally provides complete connectivity between all of the end terminals [(ET)] 110-1 through 110-N and call processors [(CP)] 120-1 through 120-N.

The end terminals [(ETs)] 110 may be embodied as any communication device in a packet network, including Internet Protocol [IP] telephones, workstations, packet telephone adapter and a facsimile machine. The call processors [(CPs)] 120 may be embodied using the call processor feature of the IP Exchangecom™ product, commercially available from Lucent Technologies, Inc., of Murray Hill, NJ, as modified herein to provide the features and functions of the present invention.

Typically, in an Internet Protocol [IP] network environment, each end terminal [(ET)] 110 is associated initially with a specific primary call processor 120. For example, as shown in FIG. 1, end terminal (ET1) 110-1 is associated with call processor (CP1) 120-1, whereas end terminal (ET5) 110-5 is associated with call processor (CP5) 120-5. Thus, whenever end terminal (ET1) 110-1 wants to place a call, a Call Set Up message is sent from end terminal (ET1) 110-1 to call processor (CP1) 120-1 for processing, in a known manner. According to the present invention, call processor (CP1) 120-1 will decide whether to process the request or to forward the request to another call processor 120 based on considerations discussed below. Generally, each call processor 120 measures the resources under its control, such as processor utilization and memory usage. Based on these resource measurements, the call processor 120 can declare congestion if sufficient resources are not available to process the call.

According to the present invention, if the call processor 120 determines that it is too congested to process the call, the call processor 120 enters an overload condition, selects an alternate call processor 120 and forwards the request to the alternate call processor 120. A given call processor [(CP)] 120 implicitly announces its overload condition to another call processor 120 by virtue of the forwarded congestion message. According to one feature of the present invention, each call processor [(CP)] 120 maintains an ordered list of call processors 120 that indicates whether or not each call processor [(CP)] 120 is overloaded. In this manner, an alternate call processor [(CP)] 120 can be selected using the ordered list of call processors 120.

In addition, the present invention will result in distributing the forwarded congestion messages among all of the available alternate call processors [(CPs)] 120 if one of the processors on the ordered list are also congested. Thus, in one implementation, the present invention utilizes a last message sent [(LMS)] flag indicating the last call processor 120 to receive a forwarded congestion message. Generally, a call processor [(CP)] 120 in an overload

condition will not forward another congestion message to a call processor [(CP)] 120 having its last message sent [(LMS)] flag set unless there are no other call processors [(CPs)] 120 available.

According to another feature of the present invention, the congested call processor 120 attaches a call processor identifier to the forwarded congestion message, indicating to the recipient call processor that the congested call processor 120 is in an overload condition. Thus, a forwarded congestion message will cause the recipient call processor 120 to set a flag, for example, in the ordered list of call processors [(CPs)] 120, indicating that the congested call processor 120 is congested. In one embodiment, each congestion flag has an associated timer that causes the flag to expire (or reset) after a predefined time interval that permits the congested to recover from the overload condition.

FIG. 2 is a schematic block diagram of an illustrative call processor [(CP)] 120. As shown in FIG. 2, the call processor [(CP)] 120 includes certain hardware components, such as a processor 210, a data storage device 220, and one or more communications ports 230. The processor 210 can be linked to each of the other listed elements, either by means of a shared data bus, or dedicated connections, as shown in FIG. 2. The communications port(s) 230 allow(s) the call processor [(CP)] 120 to communicate with all of the other network nodes 110, 120 over the network 100.

The data storage device 220 is operable to store one or more instructions, discussed further below in conjunction with FIGS. 4 and 5, which the processor 210 is operable to retrieve, interpret and execute in accordance with the present invention. In addition, as discussed further below in conjunction with FIG. 3, the data storage device 220 includes an overload control analysis table 300. Generally, the overload control analysis table 300 maintains the ordered list of call processors 120 and is utilized to select an alternate call processor 120 in the event of an overload condition.

In addition, the data storage device 220 includes an outgoing congestion evaluation process 400 and an incoming congestion evaluation process 500, discussed further below in conjunction with FIGS. 4 and 5, respectively. Generally, the outgoing congestion evaluation process 400 determines whether the call processor 120 is too congested to process a given call, and if so, enters an overload condition and forwards the request to a selected alternate call processor 120. The incoming congestion evaluation process 500 processes a forwarded

congestion message that has been received from a congested call processor 120. The incoming congestion evaluation process will set a flag indicating that the congested call processor 120 is congested.

FIG. 3 illustrates an overload control analysis table 300 that is maintained by each call processor 120. Generally, the overload control analysis table 300 maintains the ordered list of call processors 120 and is utilized to select an alternate call processor 120 in the event of an overload condition. In one embodiment, the overload control analysis table 300 maintains a plurality of records, 305 through 320, each associated with a different alternate call processor 120. For each call processor 120 identified in field 340, the overload control analysis table 300 maintains a congestion indicator (CI) in field 345 indicating whether or not the call processor [(CP)] 120 is congested. If the congestion indicator [(CI)] is equal to one, then the call processor [(CP)] 120 is congested. If the congestion indicator [(CI)] is equal to zero, then the call processor [(CP)] 120 is not congested. In addition, the overload control analysis table 300 utilizes the last message sent [(LMS)] bit in field 350 to indicate the last call processor [(CP)] 120 to receive a forwarded congestion message, as previously described. Finally, field 355 maintains a timer indicating the amount of time since a forwarded congestion message was received from the corresponding call processor [(CP)] 120.

In one implementation, the overload control analysis table 300 also maintains a total congestion indicator (TCI) bit. The <u>total congestion indicator</u> [TCI] bit is the outcome of the AND operation of all of the entries in the congestion indicator [(CI)] field 445. The <u>total congestion indicator</u> [TCI] bit indicates whether there is total congestion. If the <u>total congestion indicator</u> [TCI] bit is set to one, then all of the alternate call processors [(CPs)] 120 are congested, so the current call processor [(CP)] 120 does not go through the overload control analysis table 300 unnecessarily.

FIG. 4 is a flow chart describing an exemplary outgoing congestion evaluation process incorporating features of the present invention and employed by the call processor of FIG. 2. As previously indicated, the outgoing congestion evaluation process 400 determines whether the call processor 120 is too congested to process a given call, and if so, enters an overload condition and forwards the request to a selected alternate call processor 120.

As shown in FIG. 4, the outgoing congestion evaluation process initially performs a test during step 410 to determine if a call set up message has been received from an end terminal [(ET)] 110. The test is performed continuously or periodically during step 410 until a call set up message is received. Once a call set up message is received, the outgoing congestion evaluation process 400 performs a further test during step 415 to determine if the call processor [(CP)] 120 has sufficient resources to process the call set up message. If it is determined during step 415 that the call processor [(CP)] 120 has sufficient resources to process the call, then the call is processed during step 420 in a conventional manner, before program control terminates during step 425.

If, however, it is determined during step 415 that the call processor [(CP)] 120 does not have sufficient resources to process the call, then a test is performed during step 430 to determine if the total congestion indicator [(TCI)] flag is set. If it is determined during step 430 that the total congestion indicator [(TCI)] flag is set, then there are no alternate call processors [(CPs)] 120 available and program control terminates during step 425.

If, however, it is determined during step 430 that the total congestion indicator [(TCI)] flag is not set, then the outgoing congestion evaluation process 400 proceeds to identify an alternate call processor [(CP)] 120 in accordance with the present invention. Thus, the overload control analysis table 300 is utilized during step 440 to identify the next call processor [(CP)] 120 in the ordered list that is not overloaded and did not receive the last forwarded congestion message from the current call processor [(CP)] 120 (CI and LMS = 0)

A test is then performed during step 450 to determine if an alternate call processor 120 was identified during the previous step. If it is determined during step 450 that an alternate call processor 120 was not identified during the previous step, then the ordered list is reevaluated during step 460 without regard to the last message sent [(LMS)] flag. Program control then proceeds to step 450 and continues in the manner described above.

If, however, it is determined during step 450 that an alternate call processor 120 was identified during the previous step, then a call set up message is forwarded to the identified alternate call processor [(CP)] 120 during step 470, and the last message sent [(LMS)] flag in the overload control analysis table 300 is set to one for the selected alternate call processor [(CP)]

120. In addition, all of the remaining last message sent [LMS] bits are set to 0 during step 470. Program control then terminates during step 480.

FIG. 5 is a flow chart describing an exemplary incoming congestion evaluation process incorporating features of the present invention and employed by the call processor of FIG. 2. The incoming congestion evaluation process 500 processes a forwarded congestion message that has been received from a congested call processor 120. The incoming congestion evaluation process will set a flag indicating that the congested call processor 120 is congested.

As shown in FIG. 5, the incoming congestion evaluation process 500 initially performs a test during step 510 to determine if a forwarded call set up message has been received from a congested call processor [(CP)] 120. The test is performed continuously or periodically during step 510 until a forwarded call set up message has been received. Once a forwarded call set up message has been received, the incoming congestion evaluation process 500 will set a congestion indicator flag during step 520 in the overload control analysis table 300 for the congested call processor [(CP)] 120 (from which it has received the message). In addition, the timer is also set in the corresponding entry of the overload control analysis table 300. The timer will automatically cause the congestion indicator [(CI)] flag to expire for the congested call processor [(CP)] 120 after a period of time within which the congestion status should have been alleviated.

Thereafter, the incoming congestion evaluation process performs a test during step 530 to determine if the receiving call processor [(CP)] 120 itself has sufficient resources to process the received call set up message. If it is determined during step 530 that the receiving call processor [(CP)] 120 itself has sufficient resources to process the received call set up message, then the call is processed in a conventional manner during step 540 before program control terminates during step 560.

If, however, it is determined during step 530 that the receiving call processor [(CP)] 120 does not have sufficient resources to process the received call set up message, then the incoming congestion evaluation process executes the outgoing congestion evaluation process 400 during step 550 to identify a further alternate call processor [(CP)] 120, before program control terminates during step 560.

EXAMPLES

FIGS. 6A and 6B illustrate an overload control analysis table 300 populated with data for call processor (CP1) 120-1 at a given instant of time, before and after, respectively, detecting an overload condition and forwarding a call set up message to an alternate call processor. If call processor (CP1) 120-1 is congested and receives a Call Set Up message from any of the associated end terminals [(ETs)] 110-1 through 110-3, call processor (CP1) 120-1 will forward the message to the next available call processor [(CP)] 120. Call processor (CP1) 120-1 initially checks the <u>total congestion indicator</u> [TCI] bit in the overload control analysis table 300. If the <u>total congestion indicator</u> [TCI] bit equals zero, then at least one of the alternate call processors [(CPs)] 120 is not congested. Call processor (CP1) 120-1 then evaluates its overload control analysis table 300, and goes through the ordered list of the call processors [(CPs)] 120. If the next call processor in the list, such as call processor (CP2) 120-2, is not congested, but has its <u>last message sent</u> [LMS] bit set to 1, then the last call set up message that was forwarded by call processor (CP1) 120-1 was forwarded to call processor (CP2) 120-2. Thus, in order to balance the overflow load over all the non-congested call processors, call processor (CP1) 120-1 attempts to forward the message to another call processor 120. Call processor (CP1) 120-1 skips call processor (CP2) 120-2 and checks the status of the next call processor, such as call processor (CP3) 120-3. Since the congestion bit of call processor (CP3) 120-3 is set, then call processor (CP3) 120-3 gets skipped as well. The congestion indicator [(CI)] and last message sent [(LMS)] flag are set to 0 for the next call processor CP4, so <u>call processor</u> CP4 is a valid candidate. Thus, call processor (CP1) 120-1 forwards the call set up message to call processor (CP4) 120-4. Call processor CP1 sets the <u>last message sent</u> [LMS] bit to 1 in the CP4 row and sets the <u>last message sent</u> [LMS] bit to 0 in the CP2 row to indicate that the last forwarded message has been sent to <u>call processor</u> CP4. FIGS. 7A and 7B illustrate an overload control analysis table 300 populated with data for call processor (CP4) 120-4 at a given instant of time, before and after, respectively, receiving a forwarded call set up message from call processor (CP1) 120-1.

FIGS. 8A and 8B illustrate the use of the last message sent [(LMS)] flag bit. Assuming call processor (CP1) 120-1 is congested and it receives a call set up message from an end terminal [(ET)] 110, call processor (CP1) 120-1 will initially check the <u>total congestion indicator</u> [TCI] bit. Assuming the <u>total congestion indicator</u> [TCI] bit is zero, there is at least one

non-congested call processor. Call processor (CP1) 120-1 will identify a non-congested call processor and forward the call setup message. As shown in FIG. 8A, call processor (CP2) 120-2 is not congested, but call processor (CP1) 120-1 has already sent the last forwarded message to call processor (CP2) 120-2 (LMS=1). Call processor (CP1) 120-1 skips call processor (CP2) 120-2 and continues evaluating other call processors identified in the overload control analysis table 300. The remaining call processors are all congested (CI=1). Thus, call processor (CP1) 120-1 revisits the overload control analysis table 300 again, and forwards the message to the first non-congested call processor, irrespective of the last message sent [(LMS)] flag bit status.

It is to be understood that the embodiments and variations shown and described herein are merely illustrative of the principles of this invention and that various modifications may be implemented by those skilled in the art without departing from the scope and spirit of the invention.

1200-392.app